Programming in C/C++
Exercises set five: grammatical parsers

Christiaan Steenkist
Jaime Betancor Valado
Remco Bos

March 1, 2017

Exercise 30, calculator

We got a calculator grammar.

Code listings

Listing 1: grammar.gr

```
1  %baseclass-preinclude   cmath
2  %stype double
3
4  %token NUMBER
5  %left NEGATION
6  %left '*'
7  %left '/'
8  %left '+'
9  %left '-'
10 %left '$'
11
12 %%
13
14 input:
15         // empty
16 |
17         input line
18 ;
```

```
19
20 line:
21         '\n'
22 |
23         expr '\n'
24         {
25             std::cout << "\t" << $1 << '\n';
26         }
27 |
28   error '\n'
29 ;
30
31 expr:
32         NUMBER
33 |
34         expr '+' expr
35         {
36             $$ = $1 + $3;
37         }
38 |
39         expr '-' expr
40         {
41             $$ = $1 - $3;
42         }
43 |
44         expr '*' expr
45         {
46             $$ = $1 * $3;
47         }
48 |
49         expr '/' expr
50         {
51             $$ = $1 / $3;
52         }
53 |
54         '-' expr %prec NEGATION
55         {
56             $$ = -$2;
57         }
58 |
```

```
59    '$' expr
60    {
61        $$ = sqrt($2);
62    }
63  |
64          '(' expr ')'
65          {
66              $$ = $2;
67          }
68  ;
```

Exercise 31, conflicts

We fixed numerous conflicts.

First grammar

The first revision of the grammar solves the reduce/reduce conflicts that are caused by the `NUMBER` branches in the `expr` and `number` nonterminals. To solve this we removed the `expr` case because it is already represented in `number`.

```
1  %token NUMBER
2
3  %%
4
5  expr:
6      number
7  |
8      expr '+' expr
9  |
10     expr '-' expr
11 |
12   // empty
13 ;
14
15 number:
16     NUMBER
17 ;
```

Second grammar

In this grammar we also solved the shift/reduce conflicts which were caused because '+' and '-' have equal priority and no explicitly specified resolution for these types of conflicts. We set them to equal priority but reduce and not shift as is customary with these operators. Also to avoid very weird cases we modified the grammar a little.

```
1  %token NUMBER
2
3  %left '+' '-'
4
5  %%
6  expr:
7    exprpart
8  |
9    // empty
10 ;
11
12 exprpart:
13     number
14 |
15     exprpart '+' exprpart
16 |
17     exprpart '-' exprpart
18 ;
19
20 number:
21     NUMBER
22 ;
```

Exercise 32, priorities

We added a new priority to a grammar.

Code listings

Listing 2: grammar.gr

```
1  %token NR
```

```
2
3  %left NOTEQUAL
4  %left '+'
5  %left '*'
6  %left '^'
7  %right '-'
8
9  %%
10
11 expr:
12   NR
13 |
14   '-' expr
15 |
16   expr '+' expr
17 |
18   expr '*' expr
19 |
20   expr NOTEQUAL expr
21 |
22   expr '^' expr
23 ;
```

Exercise 35, separated lists

We fixed the list grammar so it does what we want it to, the right way this time.

Design

There are three kinds of list, plain, separated and empty. The types plain and separated have non-terminals but the empty list does not as this is not necessary. Each list can only have one token type in it which consists of `plain_(token)_seg` or `sep_(token)_seg` non-terminals that we made for each token. Tokens are reduced into the proper list `seg`ment at the start of the list. A program could maintain this structure in the grammar file, adding or removing rules for certain tokens.

Code listings

Listing 3: grammar.gr

```
1  %token WORD
```

```
2  %token INT
3  %token FLOAT
4
5  %scanner Scanner.h
6  %debug
7
8  %%
9
10 list:
11   plain
12 |
13   separated
14 |
15   // empty
16 ;
17
18 // Plain list
19
20 plain:
21   plain_word_seg
22 |
23   plain_int_seg
24 |
25   plain_float_seg
26 ;
27
28 plain_word_seg:
29   plain_word_seg WORD
30 |
31   WORD
32 ;
33
34 plain_int_seg:
35   plain_int_seg INT
36 |
37   INT
38 ;
39
40 plain_float_seg:
41   plain_float_seg FLOAT
```

```
42  |
43    FLOAT
44  ;
45
46  // Separated list
47  separated:
48    sep_word_seg WORD
49  |
50    sep_int_seg INT
51  |
52    sep_float_seg FLOAT
53  ;
54
55  sep_word_seg:
56    sep_word_seg WORD ','
57  |
58    WORD ','
59  ;
60
61  sep_int_seg:
62    sep_int_seg INT ','
63  |
64    INT ','
65  ;
66
67  sep_float_seg:
68    sep_float_seg FLOAT ','
69  |
70    FLOAT ','
71  ;
```